Intelligent I/O - I_2O Architecture

Every aspect of computer technology is feeling the demand for more processing power and higher I/O bandwidth. We've all heard of Moore's Law which states that computer processing power doubles every 18 months. Unfortunately, I/O technology has not kept pace with the speed of the processor and inevitably a bottleneck occurs, restricting the flow of data. As attempts are made to increase the I/O bandwidth we inevitably increase the number of interrupts sent to the host processor.

Why I_2O?

An interrupt occurs whenever a disk subsystem, a network interface card, or any other I/O device needs attention. In any given operation an I/O device may interrupt the processor many times. While the processor chips used today are blindingly fast at straight computational functions, they were not designed to handle interrupt duties. The answer is to allow the processor to do what it does best, manage the applications, and offload the I/O functions by implementing Intelligent I/O processing.

I_2O allows a second I/O processor to offload the tasks of the main processor, to free it up to do the real work. The second processor is the same concept as a DMA processor only better. I_2O allows requests to come in from a device on PCI destined for another device on PCI, and the request never has to go through the main processor. The I_2O processor recognizes the requests and handles it locally. It also allows requests to queue up at the I_2O processor while the main processor is working on other important tasks.

intel®

Intelligent I/O

Intelligent I/O most commonly refers to any server system that uses a processor as part of the I/O subsystem. The I/O processor performs duties that would normally be executed by the host processor, thereby reducing the host processor overhead. By giving the processor some relief, overall response time is increased as well as I/O throughput.

Since the initial implementation of intelligent subsystems, vendors have increasingly built servers with high bandwidth I/O. But as demand has increased, software developers have struggled to keep up with the multiple hardware drivers that interface to the various operating systems. The need arose for a hardware standard that would work across diverse operating systems and revisions.

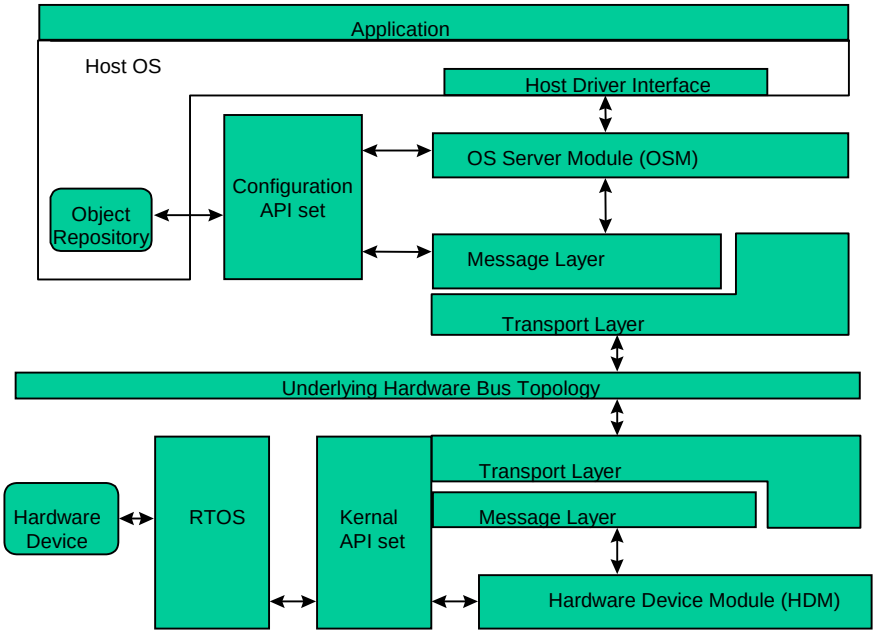
In 1996 Intel and other industry leaders formed a special interest group to address the need for a standard interface for intelligent I/O systems. The resulting standard was dubbed I_2O ("Eye Two Ohh"). The key to the I_2O architecture is that it puts a processor (or processors) between the peripheral and the operating system. Peripheral vendors are then spared the task of writing drivers for multiple operating systems. Peripheral vendors need only write one driver to the I_2O architecture and the operating system will speak to the I_2O subsystem.

The I_2O architecture also eases the task of building peripherals by simplifying the demands on the I/O cards. Much of the processing that was previously done on the card can now be done by the I_2O processor. Additionally, with a clearly defined standard the opportunity for plug and play operations becomes a reality for Intelligent I/O.

How It Works

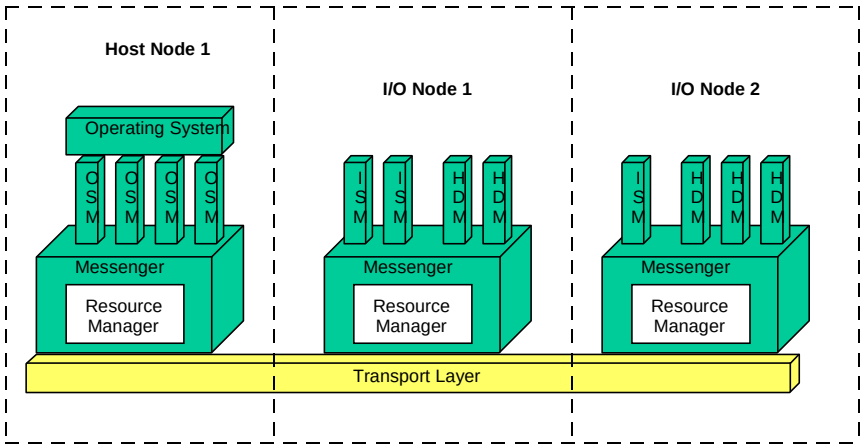
I_2O drivers are divided into two modules: The OS Services Module (OSM), and the Hardware Device Module (HDM). The OSM interfaces with the operating system and the HDM interfaces with the hardware device. The two modules exchange information through a two-layer communications system in which a message layer sets up a communications session while a transport layer defines how information will be shared. The modules communicate without knowledge of underlying bus architectures or topologies. Messages take the form of meta language, so that communications do not depend on host operating systems or bus configurations.

I^2O Execution Environment



When the OSM is presented with a request from the host operating system, it translates the request into an I_2O message and dispatches it to the appropriate HDM for processing. Upon completion of the request, the HDM dispatches the result back to the OSM by sending a message through the I_2O message layer. The OSM behaves just like any other device driver from the host operating system's perspective.

I_2O Software Architecture



The Message Layer

The foundation of I_2O is the Message Layer, which provides the glue that connects the framework of the driver model. The Message Layer is responsible for the management and dispatching of all requests. It provides a set of Application Program Interfaces (APIs) for delivering messages, along with a set of support routines that process them. APIs are language and message formats used by an application to communicate with the operating system and other system programs.

There are three basic components in the message layer: the message handle, the Message Service Routine (MSR), and the message queue. The message handle is essentially the "address" of the MSR registered in the call. A message handle is returned for every call to the Message Layer. The message queue provides the link between the requester and the desired service.

When a driver request is made, a message is deposited in a message queue and an MSR is activated to process the request. Messages themselves are made up of two parts: a header and a payload, where the header describes the types of request along with the return address of the originator.

I_2O is based on a queue between the requester and the MSR. The requester and service module can reside either on separate execution environments, or on a single processor system. I_2O also defines a neutral memory format, which provides independence from the host operating system.

The Operating System Services Module - OSM

The OSM provides the interface between the host operating system and the I_2O message layer. In the split driver module, the OSM represents the portion of the driver that interfaces to the host specific API, translating them to a neutral message format that is then sent to an HDM for processing.

The OSM translates requests from the host operating system into messages that can be dispatched to the appropriate HDM for processing. HDM information is forwarded back to host operating system through the OSM via the I2O Message Layer.

Developers can also create host OSMs that work with multiple HDMs. By implementing an OSM with a single message handle which services multiple queues from different service modules, a single OSM can send and service requests to and from multiple devices.

The Hardware Device Module - HDM

The HDM is the lowest level module in the I_2O environment, and provides the device specific portion of the driver that understands how to interface with the particular controller and devices. HDMs are roughly analogous to the hardware-specific portion of the network and SCSI drivers that exist today. The HDM translation layer is unique to each individual hardware device and vendor.

Conclusion

While performance is obviously tied to specific applications and hardware, performance gains as high as 60% have been realized by some I/O bound database procedures. That is pretty good considering the incremental costs for an I_2O equipped server run approximately 15%.

The I_2O architecture is clearly a win/win situation for all parties. The hardware vendors get a break on writing drivers, operating systems are removed from some of the low level I/O they routinely face, and customers receive substantial improvements in performance with a tremendous return on investment.

Key Points to Remember

- Intelligent I/O commonly refers to any server system that uses a processor as part of the I/O subsystem.
- I_2O offloads certain I/O operations resulting in an overall improvement in system performance.
- I_2O allows developers to write device drivers that are portable across operating systems.
- The I_2O split driver model includes the Operating System Services Module (OSM) and the Hardware Device Module (HDM).